Development of a GPS-Based Transit Tracking System for Corvallis

by

Daniel F. Urbanski

A PROJECT

submitted to

Oregon State University

University Honors College

in partial fulfillment of
the requirements for the
degree of

Honors Baccalaureate of Science in Computer Science (Honors Scholar)

Presented August 29, 2012
Commencement June 2012

AN ABSTRACT OF THE THESIS OF

Daniel F. Urbanski for the degree of Honors Baccalaureate of Science in Computer Science presented on August 29, 2012. Title: Development of a GPS-Based Transit Tracking System for Corvallis.

Abstract approved:

_____

Mike Bailey

Buses can be impractical for those who must adhere to a strict schedule or depend on them for emergencies. While variations from the official bus schedule are understandable and largely unavoidable, a lack of communication discourages adoption at a rate disproportionate with their actual likelihood. Even if a bus is running exactly on schedule, bus users have no easy way of knowing that information and those that have alternative modes of transportation will be less likely to ride the bus regardless of its actual timeliness.

The CTS BusTracker system utilizes modern technologies to provide Corvallis bus users with accurate, real-time information about the arrival times of buses within the city. The project consists of two main parts: a GPS (Global Position Satellite) tracking system that uses the bus's positional data and route information to calculate estimated arrival times, and a communication infrastructure that allows potential bus users to query and receive this information in real time.

Key Words: Android, ASP.NET, bus, GPS, MySQL, smartphones, transit, website

Corresponding e-mail address: urbanski@gmail.com

Development of a GPS-Based Transit Tracking System for Corvallis

by

Daniel F. Urbanski

A PROJECT

submitted to

Oregon State University

University Honors College

in partial fulfillment of
the requirements for the
degree of

Honors Baccalaureate of Science in Computer Science (Honors Scholar)

Presented August 29, 2012
Commencement June 2012

Honors Baccalaureate of Science in Computer Science project of Daniel F. Urbanski presented on August 29, 2012.

APPROVED:

Mentor, representing Computer Science

Committee Member, representing Computer Science

Committee Member, representing Computer Science

Chair, Department of Computer Science

Dean, University Honors College

I understand that my project will become part of the permanent collection of Oregon State University, University Honors College. My signature below authorizes release of my project to any reader upon request.

Daniel F. Urbanski, Author

Table of Contents

I. INTRODUCTION 1

Problem Description 1

Related Work 3

Project Goals 4

II. REQUIREMENTS 5

System Features 5

Software/Hardware Requirements 7

III. SYSTEM ARCHITECTURE 8

Overview 8

User Interface 8

Class Structure 10

IV. TECHNOLOGIES USED 14

Programming Languages / Software 14

Development Tools 16

Services 18

V. CONCLUSIONS 20

Challenges 20

Future Plans 23

Table of Contents (continued)

Final Thoughts 23

BIBLIOGRAPHY 25

Helpful Links 25

APPENDICES 27

Appendix A – Use Case/ Dataflow Map 27

Appendix B – Essential Code Listings 28

Appendix C – Data Structure of CT_GTFS Database 34

List of Figures

Figure 1: BusTracker Website 9

Figure 2: QR Code Containing URL with Stop ID.....9

Figure 3: Class Dataflow Map 10

Figure 4: Bus Class Diagram 10

Figure 5: Stop Class Diagram 12

List of Acronyms

ASP – Active Server Page

CTS – Corvallis Transit System

DDL – Dynamic Link Library

ETA – Estimated Time of Arrival

GPS – Global Positioning System

GTFS – General Transit Feed Service

IDE – Integrated Development Environment

MAX – Metropolitan Area Express

MSDNAA – Microsoft Developers Network Academic Alliance

OS – Operating System

QR Code – Quick Response Code

SDK – Software Development Kit

SMS – Short Message Service

SQL – Structured Query Language

TfL – Transport for London

URL – Uniform Resource Locator

I. INTRODUCTION

Problem Description

As population numbers in cities increase and gas prices rise, public transportation is often suggested as an easy, cheap, and environmentally friendly alternative to driving, but the uncertainty inherent to the system combined with a lack of communication often prevent its widespread adoption by commuters. Busses in particular can be impractical for those who must adhere to a strict schedule or depend on them for emergencies. Buses that are running particularly behind schedule can lead to late arrivals or missed connections while busses running ahead of schedule can indirectly cause travelers to be late if they end up waiting for a bus that has already passed. While these variations from the official bus schedule are understandable and largely unavoidable, the lack of communication discourages adoption at a rate disproportionate with their actual likelihood. Even if a bus is running exactly on schedule, bus users have no easy way of knowing that information and those that have alternative modes of transportation will be less likely to ride the bus regardless of its actual timeliness.

In this modern era of technological communication, it is increasingly easy for people to stay in contact at all times with the use of smartphones and other internet-capable mobile devices. While business has traditionally been conducted during specified business hours and preplanned locations, communication and scheduling software advances in recent years have made it easier to facilitate impromptu meeting or work schedule changes. Many employees in time-critical jobs and industries are given smart

phones specifically so they can stay in communication and come in to work at unusual times if necessary. Because of these advances, more precision is required in scheduling activities, and employees are expected to be able to give more accurate time estimates. These technologies have helped make businesses run more efficiently but put commuters who use public transportation at a significant disadvantage. Workers who are unable to provide employers or clients with up-to-date time arrival estimates or warn of delays in a timely manner risk endangering their careers. The lack of certainty that public transit users face potentially applies to everyone and not just commuters in the business world. Students, for instance, need to arrive on time for classes and meetings with professors and their peers, and commuters traveling to social engagements risk missing events or being seen as rude for not notifying of their tardiness.

If, however, there were a way for potential bus users to be aware of delays in real-time, it would provide reassurance for those who have other means of transportation and would help eliminate some of the uncertainty for those who completely reliant. Bus users who know ahead of time that their bus is running behind or has already been missed can potentially try to make other arrangements and secure alternative modes of transportation, rather than waste time waiting at a stop. Even if the bus user has no other means of getting to their destination, knowing about potential bus delays ahead of time allows them to prepare others for their eventual tardiness. Employees can notify their managers or business clients and possibly reschedule meetings, and students can notify their group project members to wait or start working without them.

Related Work

One of the first publically accessible transit tracking systems was London's "Countdown" service which was first implemented in 1992. This program started as a collection of 480 display signs on one bus route that provided estimated arrival-time information [1] but was later expanded to cover all of London's 19,000 stops. The notifications system proved very popular with customers, and a follow-up study found that the additional information helped reduce anxiety about whether a bus had been missed and actually made users think that the buses had a higher on-time arrival rate than their actual on-time arrival rate [2]. In 2011, Transport for London (TfL) further expanded the Countdown service to allow commuters to check arrival information online and SMS [3]. London has over 140 times the population of Corvallis, so the usage requirements for their system are quite different, but Countdown also served as the inspiration for Portland TriMet's TransitTracker service.

TransitTracker offers Portland transit users a way of tracking buses and MAX trains using satellites and sensors in the tracks [4]. Each bus or MAX stop has a unique numeric ID code that the user can use with the system. By specifying this code through a web form or SMS text, the user can receive information about the next transit vehicles to arrive at that stop. TransitTracker relies on a Bus Dispatch System (BDS) implemented by TriMet in 1997 that includes on-board Automatic Vehicle Location (AVL) devices [1]. This system, while accurate and efficient, relies on an infrastructure of specialized technologies that not all transit services have access to. A city would need to develop their own AVL to work with TransitTracker, and since these technologies are not standardized, any existing system would most likely need to be heavily modified for

compatibility. The biggest downside to implementing TransitTracker in a new city would be the lack of flexibility and the overhead costs of outfitting the buses with compatible technologies.

Project Goals

The goal of this project is to develop a system that utilizes modern, publically available technologies to provide Corvallis Transit System (CTS) bus users with accurate, real-time information about the arrival times of buses within the city. The project consists of two main parts:

1. A GPS tracking system that uses the bus's positional data and route information to calculate estimated arrival times.
2. A communication infrastructure that allows potential bus users to query and receive this information in real time.

This project serves as a proof-of-concept to show how this system can be implemented using publically available technologies with minimum cost. While more expensive and potentially more accurate GPS-based tracking solutions exist, this project uses well documented and easily accessible technologies, and therefore it will be simpler for other programmers to understand. This approach offers the key advantage of making it easier for new programmers to maintain, adapt, and expand upon this code in the future. Additionally, the project requirements were designed with budget in mind, and so the designed system has very little start-up and ongoing maintenance costs.

II. REQUIREMENTS

When working on the CTS BusTracker, it was important to outline the initial requirements for the system in order to facilitate development and evaluate performance. This project is a proof-of-concept and was mostly self-led, so there was no specific client in mind. Instead, I established these requirements early on in the development process and used them as a way to schedule my time working on the project and to check my progress towards the project goals. This section is divided into System Features, Interface Requirements, and Software/Hardware Requirements. In System Features, I outline the specific capabilities of the project from the perspectives of both the transit system operators and potential bus riders using the system dividing these into a section for the Android app and the web server, and in Software/Hardware Requirements, I outline the necessary software and hardware components required to implement the system.

System Features

Android App

The Android app's primary purpose is to facilitate the communication of the device's precise geographical location using the phone's Global Position Systems (GPS). This location is then used in calculations performed by the web server.

- The app frequently sends the GPS coordinates and compass direction to a remote database (Note: Originally, the GPS information was to be transmitted directly to the web server via TCP socket communication, but this was decided against. See Conclusion).
- The user can simply turn the transmission of the devices GPS coordinates on and off

- The user can specify a unique id number for the particular bus and device
- The user can select the current route and trip id

Web Server

The web server is in charge of keeping track of the bus GPS information, performing the calculations to estimate arrival times, and hosting the website that displays this information to the user.

- **Database**
    - The database stores the route information for the Corvallis Transit System (CTS)
    - The database contains the latest transmitted locations from the bus GPS devices along with the date and time of transmission
- The web server calculates the estimated time of arrival (ETA) for buses at a specified stop
- The website displays the ids of the next 5 buses to arrive at the given stop, their route IDs, and their ETA
- The user can scan an automatically generated QR Code to find arrival information for their specific stop
- The web site allows the user to manually select or type in their stop id and optionally select a specific bus route
- The user can bookmark the page for their particular route and stop
- The web site is formatted so that it can be clearly viewed and used from an internet-capable mobile device

Software/Hardware Requirements

Android Smartphone

- Android OS 2.3 or higher
- GPS-Enabled
- Internet connection

Windows Web Server

- Windows Server 2008 or higher
- ASP.NET installed
- MySQL installed
- PHP installed
- .NET 4.0 Framework installed

Additional Requirements

- Access to the latest CTS route information in GTFS format
- A phone network data plan for Internet access
- Administrator privileges for creating a new tables in MySQL

III. SYSTEM ARCHITECTURE

Overview

The website runs on the ASP.NET framework. Besides the main class that handles the user interface, the BusTracker system consists of two custom classes that perform most of the queries and calculations for estimating bus arrival times. In addition, I use a custom structure created to hold GPS coordinates that is based on the GeoCoordinate class that is only available in the .NET 4.5 Framework. When a user searches for the next bus arrival times from their browser, a Stop object is created for their stop ID. This Stop object creates a list of the next bus trips that will hit that stop and then uses these to create a list of Bus objects. These Bus objects are then able to perform calculations and communicate with the Google Maps API to return the ETAs that are then displayed on the main page for the user. For a more complete picture of how this information is communicated, see the **Use Case / Dataflow Map** in Appendix A.

User Interface

This website is designed to be simple for the user to enter their stop and route specification and easy for them to see the ETA information. The user input consists of two drop down lists, a textbox, and two separate submit buttons. The first drop down list is automatically filled with all the routes in CTS_GTFS database. Once the user selects a value, the other drop down list becomes available and is populated with all the stops on that particular route. After the stop is selected, the top submit button becomes enabled and the user can receive the ETA information for that route and stop combination. If the

user does not want to specify a route, they can type the stop ID manually into the textbox and click the lower submit button.

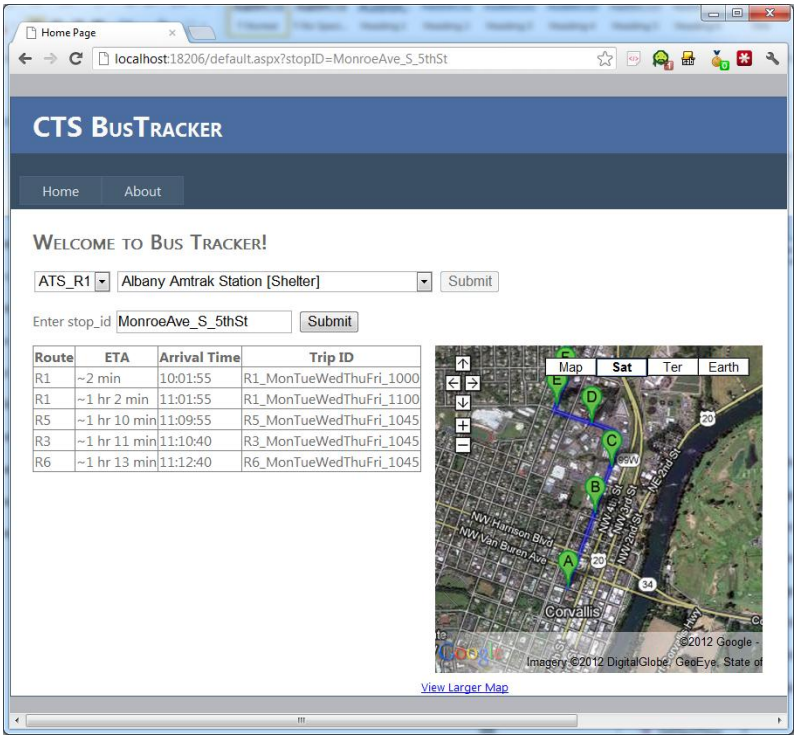


Figure 1: BusTracker Website

After the user clicks the appropriate submit button or enters the parameters directly through the URL, a table displaying the estimated arrival information for the next five buses to pass through the given stop appears. If the route ID is specified, the table only displays the information for buses on the given route. The table displays the bus routes, approximate time until arrival in minutes, estimated arrival time, and the trip id. In addition to the table, a map from Google is displayed that highlights the bus' current location in relation to the user's stop.

If the user does not want to enter the stop ID manually and has access to a smartphone with a camera, they can use a barcode scanning app to scan a unique QR code for that stop that will take them to the correct page. The site is setup so that it can detect when a mobile browser is viewing it and reset the layout accordingly so that is can be easily viewed.

Figure 2: QR Code containing URL and stop ID

Class Structure

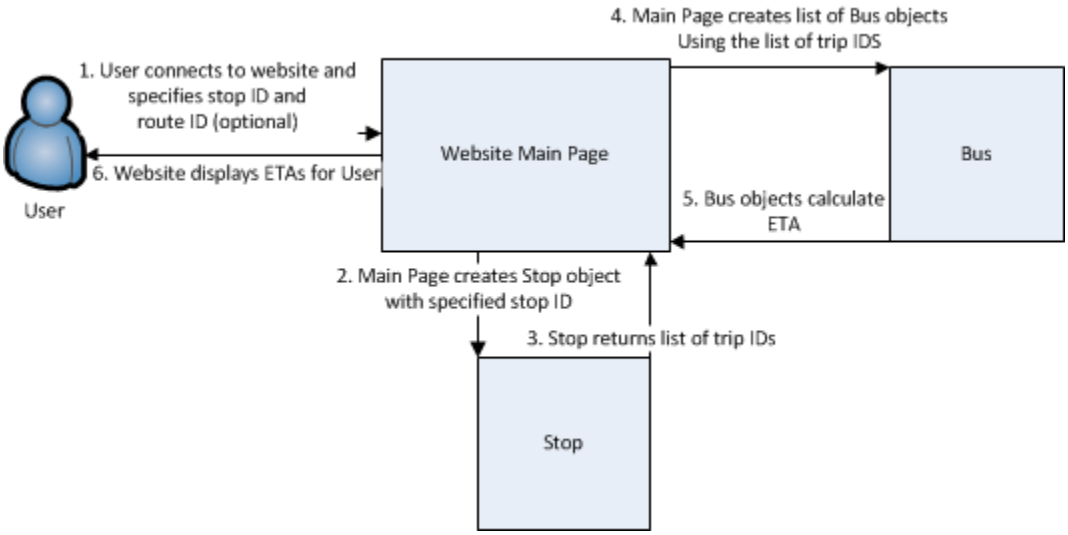


Figure 3: Class Dataflow Map

Bus Class

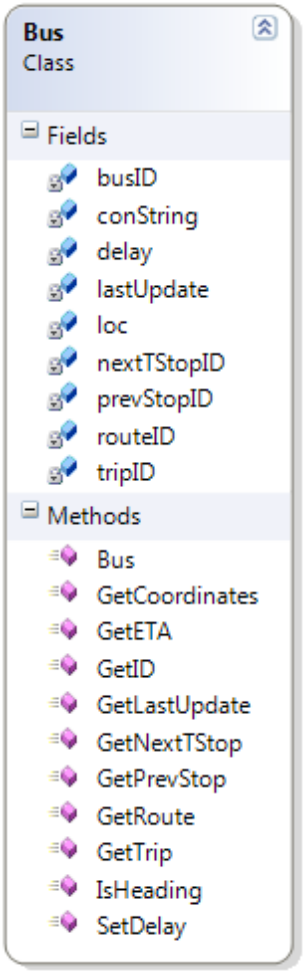


Figure 4: Bus Class Diagram

As its name implies, the Bus class represents the buses equipped with the GPS-device and BusTracker app. Each bus object stores the information for a specific bus at the time of the object's creation. Each instance contains that vehicle's geographic coordinates, bearing, current trip ID, and the time that this information was recorded. The object also contains functions that can retrieve this information and make calculations using the Google Maps API.

The constructor for the Bus class takes the connection string for the CTS_GTFS database and a trip ID as a parameter. A trip ID is a string that CTS uses to identify each embarkation of a route (e.g. "R4_MonTueWedThuFri_1045"). At any given time, each

bus on route has only one trip ID, and it is unique to that bus. After a route has been completed, the bus is assigned a new trip ID for the new departure time, even if it is repeating the same route. On the Android app, the bus driver manually selects this trip ID at the beginning of each new trip, although it could be theoretically calculated using the current route information and time of departure. When a Bus object is created, the constructor uses the specified trip ID to query the "bus" table on the MySQL database and retrieve the most recent geographic information for the bus on that trip.

One of the key methods in the Bus class is the "GetETA" function that takes a Stop object as a parameter and calculates the estimated time of arrival for a bus at the given stop. The first thing that the function does is try to estimate the delay for bus to see how close it is to its schedule. This value is an integer representing the number of milliseconds delay and can be negative if the bus is running ahead of schedule. This value is calculated by calling the Bus object's "GetNextTStop" functions to query the "bus" table for the next stop on its route that has a listed arrival time. Only some stops on a given route have listed arrival times, and these will be referred to as timed stops. The program then makes a list with the GPS coordinates for bus' current location plus all the intermediary stops to get to the next timed stop. This list is then sent to the Google Maps API to receive the travel time estimate. The travel time is then added to the current time to create an ETA for timed stop, and this is compared to the time of the scheduled arrival to calculate the delay.

If the requested stop is a timed stop, the "GetETA" function simply adds the delay to the scheduled arrival time and returns the new time. If the stop isn't a timed stop, the function sends the GPS coordinates of the last timed stop and intermediary stops to the

Google Maps API to get the travel duration. This time is then combined with the delay and then added to the scheduled arrival time of the last timed stop to get the ETA of the requested stop.

Stop Class

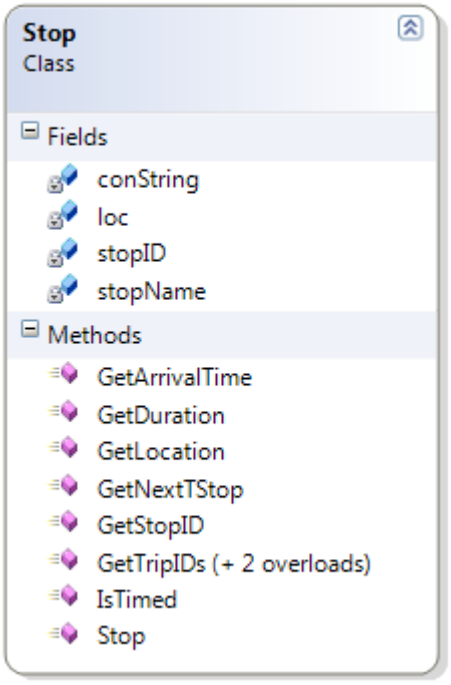


Figure 5: Stop Class Diagram

Stop objects are created when the user specifies the bus stop location for which they want to calculate the ETA. This location is conveyed through a stop ID string that is passed as a parameter in the URL of the BusTracker web page. Users can specify the stop ID by selecting it from a drop down list for their route, typing it into a textbox, entering it manually into the URL, or by scanning an automatically generated QR code for that stop. Additionally, users have the option of selecting a specific route so that only information about buses on that route will be displayed.

The constructor for the Stop class takes the connection string for the CTS_GTFS database and a stop ID as a parameter. A stop ID is a unique string assigned by CTS to represent a specific bus stop. When a Stop object is created, the constructor queries the "stop" table of CTS_GTFS database and records the id, full name, and geographical coordinates in a Stop object for later retrieval.

The Stop class also contains the "GetTripIDs" function that can be used to get the trip IDs of the next buses scheduled to reach that stop. The function can take the day of the week, the current time, the route ID, and the number of results to return as

parameters. To simplify the code, overloads were created so that the number of results is the only required parameter. Route ID is optional and if no day or time is specified, these will be defaulted to the current day of the week and time. The function uses these parameters to form a query for the CTS_GTFS that returns the trip IDs of the next buses to arrive at the specified location at the specified time. The specified time is extended by 15 minutes if the stop has a set arrival time and 7.5 minutes if it doesn't. This is done to account for buses that are running very behind schedule and these results are later filtered out if it is determined that the bus has already passed the stop when its GPS coordinates are retrieved. The "GetTripIDs" function returns a list object containing the retrieved trip IDs.

IV. TECHNOLOGIES USED

Programming Languages / Software

Android SDK / Java

Android is an open source operating system for mobile devices developed by Google. One of the key features of Android is its ability to extend the functionality of devices through new applications (or apps). These apps are developed by a large community of professional and hobbyist programmers using a customized version of Java [5]. Because of this large community, Android has a lot of documentation and support for new developers.

The Android SDK is a collection of APIs and tools that facilitate Android application development. The Android SDK Manager allows developers to install individual API components for different versions of the Android OS with relative ease. One of the key tools of the SDK is the Android emulator which allows developers to deploy and test their code on a number of virtual Android phones with different hardware specifications.

I chose to develop an Android application for this project because most modern Android smartphones contain a GPS system that can be easily accessed with the correct security permissions. Java is also a language with which I am very familiar so learning the specific quirks of Android Java seemed like a reasonable goal.

ASP.NET w/ C#

ASP.NET is Microsoft's web application framework for developing and publishing full-featured web sites and web applications. The layout for sites and web behavior can be dynamically formatted using ASP code, and the backend code can be

written in any Microsoft .NET language [6]. I chose to write this code in C# because it is the language I am most familiar with and because of its similarities to Java, which I needed to use for the Android components of the system.

Because ASP.NET needs to run on a machine with a version of Microsoft Windows, I needed to rent an online Windows Server. Attempting to host the web site from my home desktop could have posed potential security risks for my personal computer, and so this solution was safer. The setup process proved to be a relatively painless process and much more straightforward than installing the appropriate technologies on my local desktop and configuring it to be accessed from the web. I used a Microsoft technology called WebDeploy to allow me to publish the site to the remote server from Visual Studio 2010 with one click.

MySQL

MySQL is an open-source database management system that is licensed for free under the terms of the GNU General Public License. Because of its open license, MySQL is one of the most popular database management systems for both commercial and personal use [7]. Data is inserted and modified using commands sent to the database in Structure Query Language (SQL) syntax. I chose to use MySQL in particular because all students are provided with a setup MySQL database for their onid accounts. This allowed easier testing before I switched over operations to the rented Windows server.

Although Visual Studio 2010 does not support MySQL database connections out of the box, I used a driver called Connector/Net to facilitate communication between the website and the MySQL server. There are several other MySQL connection drivers available, but I chose to use Connector/Net because the library is self-contained in a .dll

file. The advantage to this is that the .dll file can be uploaded to the server running the website without having to have privileges to install new software.

PHP-GTFS-MYSQL

To import the GTFS feeds, I used some open-source software written by Steffen Martinsen called php-gtfs-mysql. This is essentially a PHP website that walks you through the steps of creating and filling tables from GTFS text files. The system was designed for the importing the data from the creator's hometown in Norway, so I had to alter the code a little to accommodate variations in the CTS's GTFS format [8]. I had to alter the structure of the tables created by the code to make various IDs strings, rather than integers. I also had to write code to remove the surrounding double quote marks from various fields of the feed and change them to single quotes, so they would be recognized by MySQL. Additionally I had to fix a bug that was causing an error when a line finished with a comma, and I also made the DateTime fields nullable. This allows so null times to be entered for stops that don't have set arrival times. Before, stops that had blank arrival times were getting set for 00:00:00, and this was causing problems with the site's calculations.

Development Tools

Eclipse Java

Eclipse is one of the most popular open-source programming language editors or IDE around today. It is primarily targeted for Java development but a number of plugins and alternative builds have been developed to allow it to be used for C++ and other programming languages. One of the most popular of these plugins in recent years allows the IDE to interface with the Android SDK. After the SDK is properly linked, the Eclipse

environment allows the user to compile and automatically upload their code to an Android phone or Android phone emulator.

Eclipse came recommended as the ideal development environment for programmers starting in mobile development by Head First Android Development and several online tutorials [9]. Eclipse was also the environment on which I first learned Java, and so the familiarity helped simplify the process of learning Android programming for the first time. Another key advantage of using Eclipse is that it is multi-platform, meaning that it can be run from Windows, Mac, and Linux operating systems. This allowed me to develop on both my Windows desktop, as well as my MacBook.

Visual Studio 2010

Visual Studio 2010 is the latest released version of Microsoft's industry-standard IDE. I was able to obtain a license to use this program for free because of a partnership the Oregon State University College of Engineering has with the MSDNAA. VS2010 offers several custom tools, layouts, and features for various .NET programming languages. Although many of these features are extraneous for my uses, and I don't mind using simple or command-line coding tools, I saw many advantages in choosing VS2010 for the development environment of the site. VS2010 offers a design preview mode that I thought would be beneficial in setting and previewing the layout of the graphical components. The suite also offers several built-in tools for communicating with databases and websites. After setting up the connection in VS2010, I was able to browse and edit the MySQL database with bus route info from within the IDE. VS2010 interfaces with ASP.NET, and using Microsoft's Web Deploy technology, I was able to compile and publish my site to the rented Windows Server with the click of a button.

Services

Google Maps API

According to its website, the Google Maps API is “a free service that lets you embed Google Maps in your freely accessible web pages or mobile apps” [10]. The API serves as a way of allowing developers to quickly access information provided by Google without having to manually go to the site and deal with the user interface. In addition, the API offers a number of advanced features that help coders get the specific information that need from a particular service. In this project, I used both the Directions Web Service and Distance Matrix Web Service to calculate the estimated travel duration between GPS coordinates.

In order to communicate with the Google Maps API, a unique string or “API key” must be set for your Google account and specified for all service requests. Google uses this information to track which applications are performing requests in order to create usage data. A limitation of relying on the API for performing calculations is that Google limits the number of requests per day for each service to 2,500. In addition, the Terms of Service stipulates that the end product must be available for free and that a Google Map image must be embedded in any pages that use one of their web services [11]. These limitations only apply to the free version of the API service, and they are waived and the request limit is upped to 100,000 requests per day if the developer pays for a business license. I wasn’t originally planning on including a map with the bus’ location on the results page, but this became the simplest way to abide by the Google terms of use.

General Transit Feed Specification

The General Transit Feed Specification (GTFS) is a common format used by public transit agencies to publish their schedule and geographic data so that it can be easily displayed and used by application developers [12]. A GTFS feed is composed of a collection of text files that each models a particular aspect of transit information: stops, routes, trips, and other schedule data. Although not all transit agencies publically post this information, it is becoming increasing common as Google uses the documents for their Google Transit service. I was fortunate because CTS has this information publically available and posted multiple lists of available GTFS data [13].

Although this format is great for publishing route information, its file-based nature does not make it ideal for quick information retrieval. I chose to store the CTS route information in a database because I knew that the information would need to be easily updatable and the site would need to be able to quickly sort and filter out the information for its calculations. Importing the GTFS files into a data structure and filtering through them to find the correct information would have been to memory intensive and not very scalable. In order to import the GTFS data into the database, I used a modified version of a tool called PHP-GTFS-MYSQL.

V. CONCLUSIONS

Challenges

One of the hardest parts of this project was deciding on specific implementation elements and actually starting to code them. Because this project was self-led, I did not have a client who was giving specific requirements. As a result, plans changed frequently in the beginning. The goal for this project was always to calculate more accurate ETAs for the bus system, but how this was accomplished changed frequently in the early stages of development.

I spent a lot time going through sample route data and attempting to form my own algorithm for calculating the ETA for stops. Initially, I relied heavily on the scheduled stop times and attempted to calculate the arrival times for the stops without times by dividing the time between stops with schedule arrival time and adding an averaged "stopping" delay. I kept trying to tweak this algorithm, but just when I thought I had it, I would find a route that completely threw everything off. I eventually realized that some of the scheduled stop times posted by CTS were completely unrealistic and I couldn't rely on them too much to provide realistic time estimations.

To get more accurate travel times, I started looking into the Google Maps API and factoring in their calculations to my own algorithm. I found that in most cases, the Google Maps duration estimate was way more accurate than the posted schedule, and so I started rewriting my algorithm to rely more on the API. Eventually, I decided on a balance that uses both the Google Maps time estimations and the bus schedule to provide a more accurate arrival time. The Google estimates are used to calculate theoretical

arrival times for stops that don't have them listed and to compare to the scheduled arrival times to work out a delay for the bus.

The Android app also posed a lot of problems in the development process. The Android SDK comes with an Android phone emulator that can be booted directly from Eclipse for testing. Unfortunately, the emulator runs quite slow and has a lot of bugs that really hindered its usefulness in the testing process. It's possible to manually set the GPS coordinates of the emulator through a window in Eclipse or by connecting to the virtual phone over telnet, but this features does not always work predictably. Even if you set the phone to have access to the emulated GPS service, it doesn't always enable it. This is a known bug in the Android development community, and the best workaround I found was to launch Google Maps every time before running the BusTracker app to force the GPS to be enabled. This extra step combined with the slow speeds of the emulator made the debugging quite slow and painful.

Another problem I ran into with the emulator was communicating with it from a different computer. Originally I was planning on having the Windows server and Android phone communicate directly over the internet using TCP Sockets. The idea was that the BusTracker app would send its geographical information only after it had been directly connected to and sent an information request. The "bus' table in the database was only used to store the Android device's assigned bus ID, trip ID, and IP address. When the user on the site wanted a time estimate, the server would look up the IP address and attempt to form a connection with that particular device and send a location request. I could connect to the app running on the emulator, when I was testing it from a temporary Java server running from the same computer, but connecting to the emulator device from

an outside computer proved to be very difficult. I spent a lot of time researching this problem and attempting to set up proxies and port forwarding so that I could communicate with the virtual device. Eventually, I managed to get a signal through but this experience left me very wary of the direct communication approach.

I originally chose to do things this way because I thought it would give more timely estimates of the bus' location, but it ended up causing a lot of headaches. In this implementation, getting access to the phone is critically important, and the system falls apart when it can't be instantly reached. Since the location information wasn't being stored in a database, if the connection didn't go through, there was no easy way to find even the most recent known location.

Additionally, the old method for communication had potential scalability issues. Although the app was multithreaded, it still had to deal with each incoming request individually. This meant that if there were a lot of requests for a particular device, the performance would slow. This wasn't really noticeable during testing, but could have definitely posed a problem for a system that was fully implemented with lots of users. I didn't want to have to rely on the Android phone's processing power to handle a large number of requests.

Once I decided to periodically publish the geographic information directly to a database, the process became much simpler and less problematic. The server was no longer communicating directly with the phone, which avoided a lot of communication and security pitfalls and also took care of the potential scalability issues. The data was actually more accurate too because I could then store the time of the last location update. If the phone lost service and temporarily stopped transmitting its coordinates, for

instance, the last known location would still be stored in the database and the calculations would use its last update time.

Future Plans

I consider this project and the proof-of-concept a total success, but a lot more work would have to be done to make this system implementable on a mass scale. Above everything else, a lot more testing would need to be done to ensure the predictions stay accurate and system holds up after an extended period of time. For the project I created a simulator that would emulate multiple buses and update the database periodically with their locations for testing purposes, but I was never able to test how the system handled multiple devices running the app in the real world. In order to bring this project to the public, I would need to go through an additional stage of testing with multiple devices running the app to ensure that it still held up. Additionally, the system would need to have improved error handling to keep users in the loop if something went wrong. The ETA algorithm itself works pretty reliably, but it can always be more fine-tuned to get better results.

Final Thoughts

This project was a great experience for me and tested my coding, researching, and scheduling abilities. I think I have grown a lot as a software engineer, since starting this project, and I have already found ways to adapt my new skills. One of the main reasons I decided on this project was to learn more about web development and database management. Although I have worked on and developed new features for ASP.NET web pages and applications in the past, this was my first time designing a site from the ground up. Through this project, I have gained a much deeper knowledge of ASP.NET, C#,

Windows Server management, MySQL databases, Android app development, and working with APIs.

The most important skills I learned from this project, however, relate to time management. My initial approach to the project was very free-form and I didn't have much of a specific plan or schedule. I would just add components, attempt to put them together and fix bugs when they arose. Because of this, there were times when things were progressing very slowly, and I found myself getting bogged down in specific design issues or bugs that later turned out to be trivial or obsoleted. As time progressed, however, I increasingly found myself turning to design tools and methods that I learned from the software engineering, senior design, and project management courses that I took in college. This was an interesting experience for me because when I was taking these classes I found creating data flow charts and project schedules time consuming and a little tedious, I found them invaluable during the later stages of development.

Overall, working on this project has been a very fulfilling experience, and I enjoy looking back on my work and seeing my idea go through the software design process. It's interesting to see my idea evolve as it went through initial planning, early designs, development, and finally a functional system. By working on this project, I have not only created something that I am proud of, I have developed many professional and personal skills that I look forward to bringing to many more projects in my future.

BIBLIOGRAPHY

[1] D. Crout, “Innovations in Public Transit,” Jan. 4, 2005,http://www4.uwm.edu/cuts/bench/tracker.pdf

[2] Transit Cooperative Research Program, “Strategies for Improved Traveler Information,” Project A-20A(1), FY 1997

[3] “London bus countdown checker website tested,” Sep. 4, 2011, http://www.bbc.co.uk/news/uk-england-london-14779558

[4] “TriMet TransitTracker,” 2012, http://trimet.org/transittracker/about.htm

[5] S. Shankland, “Google’s Android parts ways with Java industry group,” Nov. 12, 2007, http://news.cnet.com/8301-13580_3-9815495-39.html.

[6] M. Harper, “What is ASP.net?” http://www.javascriptkit.com/howto/aspnet.shtml

[7] R. Schumacher and A. Lentz "Dispelling the Myths," http://ftp.nchu.edu.tw/MySQL /tech-resources/articles/dispelling-the-myths.html.

[8] S. Martinsen, “PHP-GTFS to MySQL,” 2012, http://steffen.im/?p=13.

[9] J. Simon, Head First Android Development. Sebastopol, CA: O’Reilly Media, Inc., 2011.

[10] “Google Maps API licensing,” 2012, https://developers.google.com/maps/licensing.

[11] “Google Maps/Google Earth APIs Terms of Service,” 2012, https://developers .google.com/maps/terms.

[12] “What is GTFS?” 2012, https://developers.google.com/transit/gtfs/.

[13] “googletransitdatafeed,” 2012, http://code.google.com/p/googletransitdatafeed /wiki/PublicFeeds.

Helpful Links

The following is a collection of links to tutorials and sties that proved helpful while dealing with the above technologies.

Android General

- http://www.vogella.com/articles/Android/article.html
- http://developer.android.com/training/basics/firstapp/starting-activity.html

- http://www.brighthub.com/mobile/google-android/articles/82805.aspx

Android GPS

- http://developer.android.com/reference/android/location/Criteria.html
- http://developer.android.com/reference/android/location/LocationListener.html
- http://www.vogella.com/articles/AndroidLocationAPI/article.html

Communication

- http://android-er.blogspot.com/2011/01/simple-communication-using.html
- http://developer.android.com/tools/help/adb.html
- http://www.codeproject.com/Articles/10649/An-Introduction-to-Socket-Programming-in-NET-using

ASP.NET

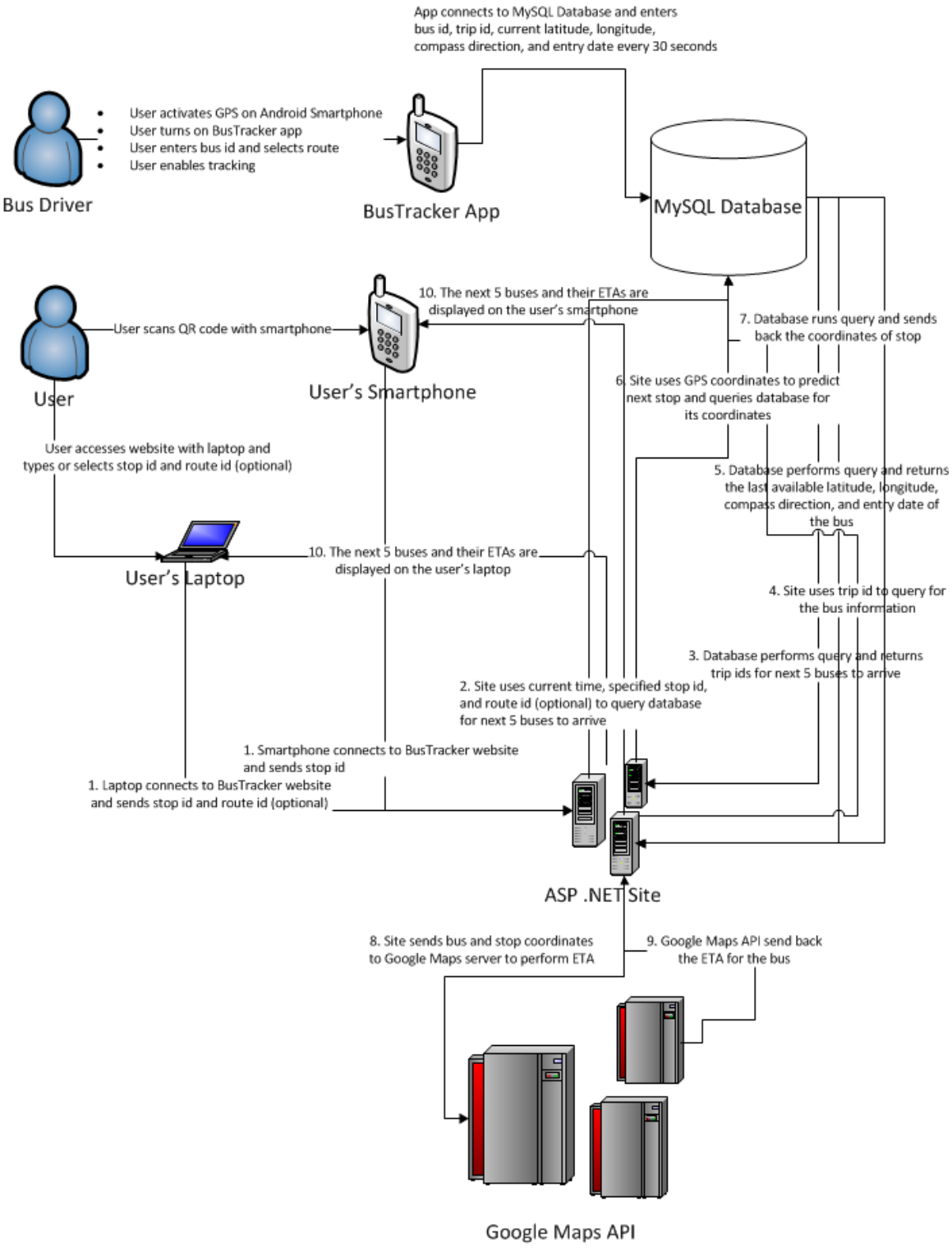
- http://www.asp.net/whitepapers/add-mobile-pages-to-your-aspnet-web-forms-mvc-application
- http://www.frederikvig.com/2009/10/creating-a-mobile-version-of-a-web-site/
- http://www.mono-project.com/ASP.NET

MySQL

- https://github.com/steffenz/php-gtfs-mysql
- http://www.connectionstrings.com/Providers/mysql-connector-net-mysqlconnection

APPENDICES

Appendix A – Use Case/ Dataflow Map



Appendix B – Essential Code Listings

Bus Class

```
public Bus(String tripID, String conString)
        {
            MySqlConnection conn = new MySqlConnection(conString);
            try
            {
                conn.Open();

                MySqlCommand cmd = new MySqlCommand();
                cmd.Connection = conn;

                cmd.CommandText =
                    @"SELECT bus_id, bus.trip_id, trips.route_id, bus_lat,
bus_lon, bus_dir, last_update FROM bus JOIN trips ON bus.trip_id = trips.trip_id
WHERE bus.trip_id = @tripID";

                cmd.Prepare();

                // Add parameters to query
                cmd.Parameters.AddWithValue("@tripID", tripID);

                MySqlDataReader queryResults = cmd.ExecuteReader();

                if (queryResults.Read())
                {
                    try
                    {
                        busID = queryResults.GetInt32(0);
                        tripID = queryResults.GetString(1);
                        routeID = queryResults.GetString(2);
                        Double tlat = queryResults.GetMySqlDecimal(3).ToDouble();
                        Double tlon = queryResults.GetMySqlDecimal(4).ToDouble();
                        loc = new GeoCoordinate(tlat, tlon);
                        busDir = queryResults.GetFloat(5);
                        lastUpdate = queryResults.GetMySqlDateTime(6).Value ;

                    }
                    catch (InvalidCastException) { }
                }
                conn.Close();
            }
            catch (MySqlException er)
            {
                throw er;
            }

        }
```

GetETA Function

```
        public DateTime GetETA(Stop stop)
        {
            Stop nextTStop = GetNextTStop();
            int derp = GetPrevStop().GetDuration(nextTStop.GetStopID(), tripID,
GetCoordinates());
```

```
            DateTime stime = nextTStop.GetArrivalTime(GetTrip());
            DateTime ttime = GetLastUpdate().AddSeconds(derp);

            int delay = (int)(ttime - stime).TotalSeconds;

            SetDelay(delay);

            Stop t_next = stop.GetNextTStop(GetTrip());

            int dur;
            if (stop.IsTimed())
                dur = 0;
            else
                dur = stop.GetDuration(t_next.GetStopID(), tripID,
stop.GetLocation());
            DateTime eta = t_next.GetArrivalTime(GetTrip());
            eta = eta.Subtract(new TimeSpan(0, 0, dur));
            eta = eta.Add(new TimeSpan(0, 0, delay));
            return eta;
        }
```

Stop Class

```
public Stop(String t_stopID, String t_conString)
        {
            conString = t_conString;
            stopID = t_stopID;
            MySqlConnection conn = new MySqlConnection(conString);
            try
            {
                conn.Open();

                MySqlCommand cmd = new MySqlCommand();
                cmd.Connection = conn;

                cmd.CommandText =
                    @"SELECT stop_id, stop_name, stop_lat, stop_lon FROM stops
WHERE stop_id = @stopID";

                cmd.Prepare();

                // Add parameters to query
                cmd.Parameters.AddWithValue("@stopID", stopID);

                MySqlDataReader queryResults = cmd.ExecuteReader();

                if (queryResults.Read())
                {
                    try
                    {
                        stopID = queryResults.GetString(0);
                        stopName = queryResults.GetString(1);
                        Double tlat = queryResults.GetMySqlDecimal(2).ToDouble();
                        Double tlon = queryResults.GetMySqlDecimal(3).ToDouble();
                        loc = new GeoCoordinate(tlat, tlon);
                    }
                    catch (InvalidCastException) { }
```

```
            }
            conn.Close();
        }
        catch (MySqlException er)
        {
            throw er;
        }
    }
```

GetTripIDs Function

```
public List<String> GetTripIDs(string routeID, int n)
    {
        List<String> tripList = new List<String>();
        String today = System.DateTime.Now.DayOfWeek.ToString().ToLower();

        // Connect to route info database
        MySqlConnection conn = new MySqlConnection(conString);
        try
        {
            conn.Open();

            MySqlCommand cmd = new MySqlCommand();
            cmd.Connection = conn;

            cmd.CommandText =
                "SELECT DISTINCT trips.trip_id AS trip, stop_times.stop_id,
stop_times.stop_sequence AS seq, " +
                "IF(stop_times.arrival_time IS NOT NULL,
ADDTIME(stop_times.arrival_time, '00:15:00'), " +
                    "ADDTIME((SELECT stop_times.arrival_time FROM stop_times "
+
                    "WHERE stop_times.trip_id = trip " +
                            "AND stop_times.arrival_time IS NOT NULL " +
                            "AND stop_times.stop_sequence > seq " +
                    "LIMIT 1), '00:07:30')) " +
            "AS latestTime " +
            "FROM trips " +
            "JOIN stop_times ON trips.trip_id = stop_times.trip_id " +
            "JOIN calendar ON trips.service_id = calendar.service_id " +
            "WHERE calendar." + today + " = true ";
            // Add route_id if specified
            if (!routeID.Equals(""))
                cmd.CommandText += "AND route_id = '" + routeID + "' ";
            cmd.CommandText +=
                "AND stop_times.stop_id = @stopID " +
                "HAVING latestTime  > CURRENT_TIME()" +
                "ORDER BY latestTime LIMIT @n";

            cmd.Prepare();
            // Add parameters to query
            cmd.Parameters.AddWithValue("@stopID", stopID);
            cmd.Parameters.AddWithValue("@n", n);

            MySqlDataReader queryResults = cmd.ExecuteReader();

            while (queryResults.Read())
            {
                try
```

```
                    {
                        tripList.Add(queryResults.GetString(0));
                    }
                    catch (InvalidCastException) { }
                }

                conn.Close();
            }
            catch (MySqlException er)
            {
                throw er;
            }
            return tripList;
        }
```

GetDuration Function

```
public int GetDuration(String tStopID, String ttripID, GeoCoordinate loc)
        {
            MySqlConnection conn = new MySqlConnection(conString);
            try
            {
                conn.Open();

                List<String> stopIDList = new List<String>();
                MySqlCommand cmd = new MySqlCommand();
                cmd.Connection = conn;
                cmd.CommandText =
                    "SELECT stop_id FROM stop_times " +
                    "WHERE trip_id = @tripID " +
                    "AND stop_sequence > (SELECT stop_sequence FROM stop_times " +
                    "WHERE stop_id = @prevStopID " +
                    "AND trip_id = @tripID LIMIT 1) " +
                    "AND stop_sequence <= (SELECT stop_sequence FROM stop_times "
+
                    "WHERE stop_id = @tStopID " +
                    "AND trip_id = @tripID LIMIT 1)";
                cmd.Prepare();
                cmd.Parameters.AddWithValue("@tripID", ttripID);
                cmd.Parameters.AddWithValue("@prevStopID", stopID);
                cmd.Parameters.AddWithValue("@tStopID", tStopID);

                MySqlDataReader queryResults = cmd.ExecuteReader();

                while (queryResults.Read())
                {
                    try
                    {
                        stopIDList.Add(queryResults.GetString(0));
                    }
                    catch (InvalidCastException er)
                    {
                        throw er;
                    }
                }
                List<Stop> stopList = new List<Stop>();

                // Add stops to list
                foreach (String sID in stopIDList)
```

```
                {
                    stopList.Add(new Stop(sID, conString));
                }

                // Form Google Maps API string

                // Add waypoints to array
                String[] wp = new String[stopList.Count - 1];
                for (int i = 0; i < stopList.Count - 1; i++)
                {
                    wp[i] = stopList[i].GetLocation().ToString();
                }

                DirectionsRequest directionsRequest = new DirectionsRequest()
                {
                    Origin = loc.ToString(),
                    Destination = stopList[stopList.Count -
1].GetLocation().ToString(),
                    Waypoints = wp,
                    Sensor = false
                };

                DirectionsResponse directions =
MapsAPI.GetDirections(directionsRequest);
                double sumDur = directions.Routes.First().Legs.Sum<Leg>(dur =>
dur.Duration.Value.TotalSeconds);

                conn.Close();

                return (int)sumDur;
            }
            catch (MySqlException er)
            {
                conn.Close();
                throw er;
            }
        }
```

GeoCoordinate Struct

```
    public struct GeoCoordinate
    {
        private readonly double latitude;
        private readonly double longitude;

        public double Latitude { get { return latitude; } }
        public double Longitude { get { return longitude; } }

        public GeoCoordinate(double latitude, double longitude)
        {
            this.latitude = latitude;
            this.longitude = longitude;
        }

        public override string ToString()
        {
            return string.Format("{0},{1}", Latitude, Longitude);
        }
```

```
        public override bool Equals(Object other)
        {
            return other is GeoCoordinate && Equals((GeoCoordinate)other);
        }

        public bool Equals(GeoCoordinate other)
        {
            return Latitude == other.Latitude && Longitude == other.Longitude;
        }

    }
}
```

Appendix C – Data Structure of CT_GTFS Database

Table structure for table agency

Column	Type	Null	Default
agency_id	varchar(55)	No	
agency_name	varchar(255)	Yes	NULL
agency_url	varchar(255)	Yes	NULL
agency_timezone	varchar(255)	Yes	NULL
agency_lang	varchar(255)	Yes	NULL
agency_phone	varchar(255)	Yes	NULL

Table structure for table bus

Column	Type	Null	Default
bus_id	varchar(55)	No	
trip_id	varchar(55)	Yes	NULL
bus_lat	decimal(13,10)	Yes	NULL
bus_lon	decimal(13,10)	Yes	NULL
prev_stop	varchar(55)	Yes	NULL
next_tstop	varchar(55)	Yes	NULL
last_update	timestamp	NO	

Table structure for table calendar

Column	Type	Null	Default
service_id	varchar(55)	Yes	NULL
monday	tinyint(1)	Yes	NULL
tuesday	tinyint(1)	Yes	NULL
wednesday	tinyint(1)	Yes	NULL
thursday	tinyint(1)	Yes	NULL
friday	tinyint(1)	Yes	NULL
saturday	tinyint(1)	Yes	NULL
sunday	tinyint(1)	Yes	NULL
start_date	varchar(255)	Yes	NULL
end_date	varchar(255)	Yes	NULL
start_date_timestamp	int(11)	Yes	NULL
end_date_timestamp	int(11)	Yes	NULL

Table structure for table calendar_dates

Column	Type	Null	Default
service_id	varchar(55)	Yes	NULL
date	varchar(255)	Yes	NULL
date_timestamp	int(11)	Yes	NULL

exception_type int(2) Yes NULL

Table structure for table routes

Column	Type	Null	Default
route_id	varchar(55)	No	
agency_id	varchar(255)	Yes	NULL
route_short_name	varchar(255)	Yes	NULL
route_long_name	varchar(255)	Yes	NULL
route_type	int(2)	Yes	NULL
route_text_color	varchar(255)	Yes	NULL
route_color	varchar(255)	Yes	NULL
route_url	varchar(255)	Yes	NULL
route_desc	varchar(255)	Yes	NULL

Table structure for table stops

Column	Type	Null	Default
stop_id	varchar(55)	No	
stop_code	varchar(255)	Yes	NULL
stop_name	varchar(255)	Yes	NULL
stop_desc	varchar(255)	Yes	NULL
stop_lat	decimal(13,10)	Yes	NULL
stop_lon	decimal(13,10)	Yes	NULL
zone_id	int(11)	Yes	NULL
stop_url	varchar(255)	Yes	NULL
location_type	int(2)	Yes	NULL
parent_station	int(11)	Yes	NULL

Table structure for table stop_times

Column	Type	Null	Default
trip_id	varchar(55)	Yes	NULL
arrival_time	time	Yes	NULL
arrival_time_seconds	int(11)	Yes	NULL
departure_time	time	Yes	NULL
departure_time_seconds	int(11)	Yes	NULL
stop_id	varchar(55)	Yes	NULL
stop_sequence	int(11)	Yes	NULL
stop_headsign	varchar(255)	Yes	NULL
pickup_type	int(2)	Yes	NULL
drop_off_type	int(2)	Yes	NULL
shape_dist_traveled	varchar(255)	Yes	NULL

Table structure for table trips

Column	Type	Null	Default
route_id	varchar(55)	Yes	NULL
service_id	varchar(255)	Yes	NULL
trip_id	varchar(55)	No	
trip_headsign	varchar(255)	Yes	NULL
trip_short_name	varchar(255)	Yes	NULL
direction_id	tinyint(1)	Yes	NULL
block_id	int(11)	Yes	NULL
shape_id	int(11)	Yes	NULL